Recommendations for the LArSoft code analysis process

Christopher Jones, Jim Kowalkowski, Rob Kutschke, Marc Paterno, Gianluca Petrillo, Erica Snider, Dorota Stefan, Robert Sulej

Version 1
May 17, 2016

1. Introduction

Regular code reviews form an essential part of any software quality assurance program. LArSoft is introducing a regime of code analyses and fostering a shift in culture that seeks third party reviews with the ultimate goal of forming a stronger foundation and design for building increasingly sophisticated algorithms via compliance with LArSoft and art design principles and good software engineering and low-level coding practices across the entire LArSoft suite.

This document reports on the recommendations for the LArSoft code analysis and review process. The recommendations are based upon a trial analysis of the Projection Matching Algorithm (PMA) code, authored by Dorota Stefan and Robert Sulej, by the authors of this document, and are further informed from the experience of the analysis team members. While the recommendations are not intended to be entirely prescriptive, they do provide considerable detail that should be a part of the review process. The ultimate responsibility for the actual process lays with the core LArSoft team leadership.

In the next section, we will describe some observations from the PMA code analysis that inform some of the recommendations, as well as other factors considered important based on previous experience. In Sects. 3 and 4, we list the types of reviews that are possible and the roles of the various participants in the analyses. This is followed in Sect. 5 with a detailed description of the various stages of the analysis process and the tasks or considerations that should be included in each. The current set of tools that can be used either to provide technical input to the analysis or support the collaborative aspects of the analysis are discussed in Sect. 6. Finally, in Sect. 7, we conclude with recommendations for engaging the community in supporting and improving the analysis process.

2. Observations from previous reviews

The PMA code analysis made a number of observations during the process of analyzing that code that bear on the recommendations for future reviews. Several team members also had considerable experience with code reviews, which was brought to the table. Some relevant observations from that previous experience also informs the recommendations, and is included below.

- The involvement and cooperation of the author (or the active maintainer of the code in question) with the process is critical to the success of reviews.
- Several types of reviews are possible. It is therefore extremely important to have a clear charge and a well-defined scope for each review.
    - While defining the charge is critical, it is also important to seek input from the code authors as to any objectives they might have in undergoing the process. The review charge should be amended accordingly in order to ensure full engagement of the author and encourage the community in further reviews.
- A specific set of preliminary work needs to be defined for each review, and should be tailored to the type of review requested.
    - Profiling results collected prior to review meetings can be performed by the code authors with the assistance of a single expert from the analysis team. This helps prevent the larger team from spending time on things that are easy to fix.
- An overview of the code presented by the authors prior to the review meetings or at the beginning of review meetings is extremely helpful in reading the code.
    - It is helpful to know what parts of the code have stabilized, and what is still in flux.

- The review meetings themselves seem to profit from a number of things
    - Several reviewers and the authors actively analyzing as a group on the same code elements
        - There is a balance to be made between having enough people to stimulate discussion and ideas, but not too many so as to make decision making overly difficult.
        - While some believed the PMA review had too many people at various points, others found the group dynamic to be helpful in generating ideas

        and identifying issues. There was also some working in parallel during the meetings, so what appeared as lack of engagement might have been partly people working on multiple fronts at any given time.
        - Some noted that design reviews might suffice with fewer people than a performance review.
    - Adequate time allocated to the review process. The team felt they reached peak productivity after about three hours in a single session. Another hour would have been helpful.
    - While narrowing focus on a few issues was critical in defining recommended changes, it was important to document other issues identified opportunistically that merit further thought or discussion after the review meetings.
    - Some thought that the person making the charge should not also lead the review, although there was not agreement on this point.
    - Meeting duration
        - For a design review, half-day blocks are the minimum useful time.
        - For a performance review, breaks during which profiling jobs are run are often necessary. Analysis sessions need considerable time blocks.
- While full review meetings are important, it is useful to consider other settings and formats than that discussed here.
    - It is important to review code as it comes into the repository. Code compliance and C++ best practice reviews can be done on immature code with considerably less effort than that required for a full review, and can head off some major problems before they become embedded.
    - Reviews in other less formal settings such as software meetings or collaboration meetings might in general be less intimidating and help to foster a culture of code reviews.
- Although developers may find profiling tools are relatively easy to use, experts will often find issues that more inexperienced users will not immediately see. This makes a review setting an ideal place to use such tools both for the analysis of the code, and as a learning experience for code authors.
- The preparatory work required a significant time investment from each of the team members -- approximately one day for each of the PMA team members. Follow-up consulting or documentation is expected to consume addition time. This needs to be taken into account and charged to the review when planning and scheduling reviews.

3. Types of reviews

We have identified the following general types of reviews that differ in the target, and therefore in the preparatory work needed, the tools used, and the level of reading of the code that might be expected.

- General review of the code design, class structure, interfaces, etc. (as compared to LArSoft and more general design principles)
- Analysis of computing performance, identification of resource bottlenecks and their solution
- Compliance with coding practices and styles (as defined by LArSoft and more general guidelines)
- Compliance with C++ best practices
- Physics performance review
    - In general, such reviews are left to the experiments, and are not considered in this document

Although a single "review" may include more than one of the above, it should be understood that each additional type of review may require added preparatory work, experts and time during the review. A light-weight process should focus on a single type at a time. The specific type should be agreed upon in advance by either the code authors the experiment offline management, and the core LArSoft team.

4. Participants and audience for a review

Each review should involve at a minimum the following people:

- Code author(s)
- An additional problem domain expert
- Two software engineering experts

Additional people can be added as deemed necessary or helpful. In general, the process should be completely open, so observers are welcome.

Each review meeting needs people in the following roles to be present:

- A scribe who documents important conclusions or points of discussion
    - Need not be the same person throughout the meeting(s).
- Domain experts
- Programming experts
- Review leader
    - Should be one of the experts, but not the scribe or the author(s).

The audience for the results of a review can include several entities, depending upon the targets identified:

- Code author(s) and the entity requesting the review
- Core LArSoft team
- Interested parties in the LArSoft community
- Experiment offline coordinators
- *art* framework team
- SCD management (in some cases)

5. Conduct of a review

5.1. Initiating a review

It should be possible to initiate a review, specify a target and define a charge based on a request from any of the following:

- The code author(s)
- An experiment representative
- The core LArSoft team
- A standing LArSoft policy (e.g., for each "pull request" equivalent)

The type of review and the scope of code involved should be decided at the time of the request in consultation with the requestor, the author, and the core LArSoft team. Whenever possible, the specific charge, review objectives and any metrics of success should be defined and agreed upon at this time.

While the criteria for the selected code is left to the experiments and the core LArSoft team, we suggest several considerations be included. Is the code a good candidate for production processing? Is there a known problem or set of issues with the code? Will improvements in performance or design have an impact beyond the code in question? Is the code a likely exemplar for other developers, or can it be used as such? Is the code mature? What is the trade-off between improving more mature code (which is therefore less likely to be re-visited to fix problems), versus trying to head off issues early in the development of code where the value to the experiments may be less obvious? Most importantly, are the code authors or relevant offline organizations supportive of the review for the code in question?

5.2. Preparatory work

The precise preparatory work will depend at some level on the type of review. Most reviews, however, will require all of the following in advance of any formal review meetings.

- Selection of the domain and software engineering experts to be involved in the review, and agreements for the time to be committed
- Selection of the people who will implement the recommended changes (can be the authors) and perform other follow-up work, and agreements for the time to be committed.
- Commonly accessible work areas in which to build and profile code
- Access to the computing resources needed by all people involved in the review
- A set of representative configurations and input data from the authors to allow the code to be run
- Aids in viewing the code during the review, such as diagrams, cloned repositories, etc.
- Reports from a static analysis tool
- Output from preliminary time and memory use profiling
    - An initial examination of the output prior to the meetings can identify and possibly address easy to fix issues early in the process.
    - The results can also be used to help understand the control flow of the code for a design review.
- If changes are to be made, a set of tests to validate changes
    - Typically tests for bitwise identical output are the most useful, since physics validation and changes that result in changes to the physics are outside the scope of these reviews
- To the extent feasible, each expert should preview of the code on his or her own.

This preparatory work can take a significant amount of time from some people, which must be included in the cost of the review and the time commitments required.

5.3. Review meetings

The structure of the review meetings should be tailored to meet the needs of the given review. We expect, however, that meetings will broadly perform or cover the following items.

- A review of the charge and discussion of special interests of the authors or review requestors.
- A discussion of how to divide the time between the various topic areas and code elements to be addressed
- A overview presentation or discussion by the authors of the algorithm and functioning of the code to be reviewed.
    - Any major concept and abstractions used by the code should be included in this.
    - The intent of this presentation is to discuss what the code does, so need not go into details such as the class structure, for instance.
- Analysis of the code. How this is done is up to those participating in the review.

Previous experience suggests that sessions of about half a day in duration are the most effective. Multiple such sessions can be organized as needed. Care should be exercised in scheduling and throughout the planning, however, so as to keep the entire process as light-weight as possible. Single session reviews, for instance, should possibly be the norm. In all cases, the time to be committed should be agreed upon in advance.

While the focus of these meetings will be confined to the pre-defined charge and scope, most reviews should allow brief discussion regarding topics of opportunity noted by the participants during the course of the review (for example, noting a use of call-by-value function arguments when `const` references would be substantially more efficient).

5.4. Review outcome and report

The direct product of the preparatory work and review meetings should include the following:

- A written report of the conclusions from the review. The report should list:
    - Recommended changes;
    - Problems or other issues that require further investigation, thought or discussion;
    - Conclusions regarding any targets of opportunity discussed;

    - Any findings that are not reflected among the items above, including discussion of potential trade-offs in any areas addressed by the review;
    - The metrics of success for the review, when they exist;
    - Any comments on the process offered.
    - A reference to comments in GitHub can be included in any of the relevant sections.

  A template for the review report might substantially reduce the time required for completion, since it could be partly or fully filled in during the review. Pre-defined sections also serve as reminders of what needs to be covered and what observations need to be recorded. Different types of reviews might need different types of templates.

- (Encouraged, but optional) Annotations to the code indicating recommended changes, or issues to investigate or discuss further.

5.5. Follow-up work

It is important to note that all of the direct benefits from a review generally accrue only after implementation of recommended changes and related improvements. We therefore stress the point that a strong and directed follow-up effort from a review is as critical to the success of the overall review as are the steps leading to the report itself. We assume that the primary responsibility for implementing changes falls to the author or experiment that owns the code. We further assume that this work will be carried out in consultation with the experts participating in the review, the core LArSoft team, and the offline management of the relevant experiment.

The details of the follow-up work, as for the review meetings, should be tailored to the particular review, the types of recommendations, and the effort available. The following tasks will ensure that the follow-up work is properly specified and tracked, and that any more generally useful conclusions will be disseminated.

- Code authors define the scope of follow-up work and create a prioritized task list in consultation with the core LArSoft team and experiment offline management.
- A set of issues entered into the LArSoft issue tracker for each of the tasks to be completed. It is expected that experiments may also choose to independently track issues for which they are responsible.

- A set of milestones entered into the LArSoft issue tracker that describe overall targets for the follow-up work. Each of the tasks should be under one of the milestones.
- Regular reports on the progress toward these milestones given at an appropriate meeting (experiment or LArSoft).
- Reporting of lessons learned in a document available to the LArSoft community.
    - All such lessons learned should be accumulated in a single location.
    - A brief report of such lessons learned may be appropriate in cases when they are or broad interest.

6. Methods and tools

The following tools and techniques have demonstrated utility in previous reviews, and should be considered for any LArSoft review, and utilized as needed.

- GitHub.com
    - Useful for providing isolated, common repositories for review participants
    - Also provides some collaboration tools that are useful, such as line-by-line annotations.
- Performance profilers
    - The IgProf profiler (http://igprof.org) is highly recommended for it's ease of use and clear output.
    - Valgrind (http://valgrind.org), a long time staple of memory profiling, is also useful, although the output tends to be more difficult to interpret.
- Static code analyzers
    - The clang static analyzer (http://clang-analyzer.llvm.org) has been used by CMS and is known to work at Fermilab. It is available under the BSD license, so can be used for reviews as needed.
- dOxygen documentation system for LArSoft
    - http://nusoft.fnal.gov/larsoft/doxsvn/html/index.html
- Tools for making class structure and other software engineering diagrams.
- Validation tools (already discussed above)
- Report templates (already discussed above)

7. Marketing and improving the process

The goal of changing the culture to accept reviews as a natural part of the development process will require a marketing campaign of sorts. No doubt some of the reticence to participate in such reviews stems from a perception that they are either overly critical of code authors or their work, that they impose a cost to "fix" or cosmetically re-structure code that already "works", or that the payoff is simply too small to make the effort worthwhile.

We suggest a multi-pronged approach to challenge these ideas and reverse these or other negative perceptions. The first is to disseminate information at both LArSoft and experiment meeting (e.g., collaboration meetings), and advertise cases in which reviews have resulted in concrete improvements. A cost accounting for the improvements should also be included, so as to demonstrate that there is a high return on investment in the review process. Ideally, these facts should be presented by the code authors themselves or the experiment offline management. If we are not making authors or the experiments glad that they participated, then we are missing the mark.

Second, the process must be completely transparent and responsive to the needs of the community. Review targets and suggested changes must have a clear benefit. Input from the authors and the community should be incorporated at all stages of the process.

Finally, we should use lessons learned, documented design principles, etc., to help developers write better code from the outset. There are numerous examples where following a simple set of guidelines would have made the code significantly better before reaching a review, which in turn makes the reviews easier to conduct, or better, allows more subtle issues to be addressed.